SML.NET: Functional programming in a non-functional world

Andrew Kennedy
(with Nick Benton & Claudio Russo)

Microsoft Research Cambridge

MSR Cambridge Programming Principles and Tools Group

- Luca Cardelli

- Nick Benton
- Cedric Fournet
- Andy Gordon
- Tony Hoare
- Andrew Kennedy
- Simon Peyton Jones
- Simon Marlow
- Claudio Russo
- Don Syme
- + visitors, PhD students, interns, ...

Overview of talk

- Introduction
  - What is SML?
  - What is SML.NET?
- Language interop in SML.NET
- Demo
- Implementation
- Visual Studio integration

Advanced Programming Languages on the CLR

- .NET provides a great opportunity for programming language designers and implementers:
  - The CLR provides services (execution engine, garbage collection, …) which make producing a good implementation of your language easier
  - The frameworks & libraries mean you can actually do useful things with your new language (graphics, networking, database access, web services,…)
  - Multi-language component-based programming makes it much more practical for other people to use your language in their own projects

What is SML?

- ML = **MetaLanguage**
  - Originally for automated theorem proving
  - Now general purpose language
- Only "real" language with a complete formal semantics
- Hugely influential in the programming language research community and widely used in teaching
- Two major variants: Caml/O'Caml (from INRIA, France) and Standard ML (most recent revision 1997)
- Applied to network stacks, distributed application toolkit, file synchronizer, web browser, server-side scripting, compilers, theorem provers

Introduction to SML in three slides

- Focus on three characteristic features:
  - Datatypes and pattern matching
  - First-class functions
  - Parametric polymorphism
- These turn out to be the challenges for implementation…

Datatypes and pattern matching

```
datatype Tree =
  Leaf
| Branch of { key:string, data:int, left:Tree, right:Tree }



fun find (tree, mykey) =
case tree of
  Leaf => raise NotFound
| Branch { key, data, left, right } =>
  case String.compare(mykey,key) of
    LESS => find (left, mykey)
  | GREATER => find (right, mykey)
  | EQUAL => data
```

Datatypes and pattern matching

```
datatype Tree =
  Leaf
| Branch of { key:string, data:int, left:Tree, right:Tree }

fun find (tree, mykey) =
case tree of
  Leaf => raise NotFound
| Branch { key, data, left, right } =>
  case String.compare(mykey,key) of
    LESS => find (left, mykey)
  | GREATER => find (right, mykey)
  | EQUAL => data
```

A tree is *either* a Leaf...

...*or* it's a branch with key, data and two subtrees

Now we can define a search function that deconstructs the tree using *patterns*

Datatypes and pattern matching

```
datatype Tree =
  Leaf
| Branch of { key:string, data:int, left:Tree, right:Tree }

fun find (tree, mykey) =
case tree of
  Leaf => raise NotFound
| Branch { key, data, left, right } =>
  case String.compare(mykey,key) of
    LESS => find (left, mykey)
  | GREATER => find (right, mykey)
  | EQUAL => data
```

A tree is *either* a Leaf...

...*or* it's a branch with key, data and two subtrees

Now we can define a search function that deconstructs the tree using *patterns*

find has inferred type
Tree * string -> int

First-class functions

```
fun find (tree, mykey, compare) =
case tree of
  Leaf => raise NotFound
| Branch { key, data, left, right } =>
  case compare (mykey, key) of
    LESS => find (left, mykey, compare)
  | GREATER => find (right, mykey, compare)
  | EQUAL => data

...find (mytree, "dOnTcArE",
  fn (x,y) => String.compare(toUpper x, toUpper y))...
```

Parameterize on the comparison function used

find now has type
Tree * string * (string*string->order) -> int

An anonymous function

Parametric polymorphism

Types can be parameterized
Here there are two type parameters

```
datatype ('k,'d) Tree =
  Leaf
| Branch of { key:'k, data:'d, left : ('k,'d) Tree, right : ('k,'d) Tree }


fun find (tree, mykey, compare) = ...
```

Code for find is the same but has type
'(k,'d) Tree * 'k * ('k*'k->order) -> 'd

What is SML.NET?

- Compiler for SML that targets verifiable CIL
- Research Issues:
  - Can we compile a polymorphic functional language to a monomorphic, object-oriented runtime?
    - Yes. The whole of Standard ML is supported.
  - How can we make it produce fast, compact code?
    - Whole-program optimising compiler. Monomorphisation. Representation tricks. Novel typed intermediate language using monads to track side-effects.
  - How can we make cross-language working easy?
    - We extend SML to support all of the .NET CLS (Common Language Specification), providing smooth bidirectional interoperability with .NET framework libraries & other .NET languages
- Visual Studio integration in progress

Is SML.NET available yet?

- Yes. Find it from
  http://research.microsoft.com/projects/sml.net
- Requirements:
  - .NET Framework Redist, *or*
  - .NET Framework SDK, *or*
  - Visual Studio .NET
- Distribution contains
  - binaries
  - documentation
  - demos
  - (optional) sources: requires SML/NJ to build.

Overview of talk

- Introduction
  - What is SML?
  - What is SML.NET?
- Language interop in SML.NET
- Demo
- Implementation
- Visual Studio integration

Previous approaches to interop

1. Bilateral interface with marshalling and explicit calling conventions (e.g. JNI, O'Caml interface for C).
   - Awkward, ugly, tied to one implementation, only for experts
2. Multilateral interface with IDL (e.g. COM, CORBA) together with particular language mappings (e.g. H/Direct, Caml COM, MCORBA).
   - Have to write IDL and use tools, tend to be lowest-common denominator, memory management often tricky

Interop in .NET

- Languages share a common higher-level infrastructure (CLR):
  - shared heap means no tricky cross-heap pointers (cf reference counting in COM)
  - shared type system means no marshalling (cf string<->char* marshalling for Java<->C)
  - shared exception model supports cross-language exception handling

SML.NET interop

- Sounds great, but SML is not object-oriented
  - So we are going to have to do *some* work…
- Possible approaches to interop:
  - do not extend language; instead provide wrappers that give a functional view of a CLS library (Haskell, Mercury).
    - Powerful functional type systems can go a very long way towards modelling OO type systems
  - redesign the language (OO-SML?)
- Our approach – a middle way (embrace and extend ;-)
  - *embrace* existing features where appropriate (non-object-oriented subset)
  - *extend* language for convenient interop when "fit" is bad (object-oriented features)
  - *live with* the CLS at boundaries: don't try to export complex ML types to other languages (what would they do with them?)

Extract from Windows.Forms interop

```
open System.Windows.Forms System.Drawing System.ComponentModel

fun selectXML () =
  let
    val fileDialog = openFileDialog()
  in
    fileDialog.#set_DefaultExt("XML");
    fileDialog.#set_Filter("XML files (*.xml) |*.xml");
    if fileDialog.#ShowDialog() = DialogResult.OK
    then
    case fileDialog.#get_FileName() of
      NONE => ()
    | SOME name =>
      replaceTree (ReadXML.make name, "XML file '" ^ name ^ "'")
    else ()
  end
```

no args = ML unit value

CLS Namespace = ML structure

constructor = ML function

static constant field = ML value

CLS string = ML string

null value = NONE

NEW!
instance method invocation

Overview of talk

- Introduction
  - What is SML?
  - What is SML.NET?
- Language interop in SML.NET
- Demo
- Implementation
- Visual Studio integration

Ray tracing in ML

- ICFP programming competition: build a ray tracer in under 3 days
- 39 entries in all kinds of languages:
  - C, C++, Clean, Dylan, Eiffel, Haskell, Java, Mercury, ML, Perl, Python, Scheme, Smalltalk
- ML (Caml) was in 1st and 2nd place

Ray tracing in SML.NET

- Translate winning entry to SML
- Add Windows.Forms interop
- Run on .NET CLR
- Performance on this example twice as good as popular optimizing native compiler for SML
  - (though on others we're twice as bad)

Ray tracing in SML.NET

- Translate winning entry to SML
- Add Windows.Forms interop
- Run on .NET CLR
- Performance on this example twice as good as popular optimizing native compiler for SML
  - (though on others we're twice as bad)

Compiling functional languages

- The CLR isn't what a functional programmer would have designed, but:
  - it has tail call support
  - it is *possible* to compile ML
  - performance results are good, considering
- Challenges (= desirable runtime features):
  - datatypes
  - first-class functions
  - parametric polymorphism
- And for Haskell:
  - laziness (thunks, shortable indirections)

Compiling ML tuples

- ML tuple and record types map to CLR classes with immutable fields for the components of the tuple. Examples:

```
type intpair = int*string
type pairpair = intpair*intpair
```

```
class IP
{  int v1;  string v2 }
```

```
class PP
{  IP v1; IP v2; }
```

- Tuples are allocated on the heap. Therefore try to flatten where possible e.g.

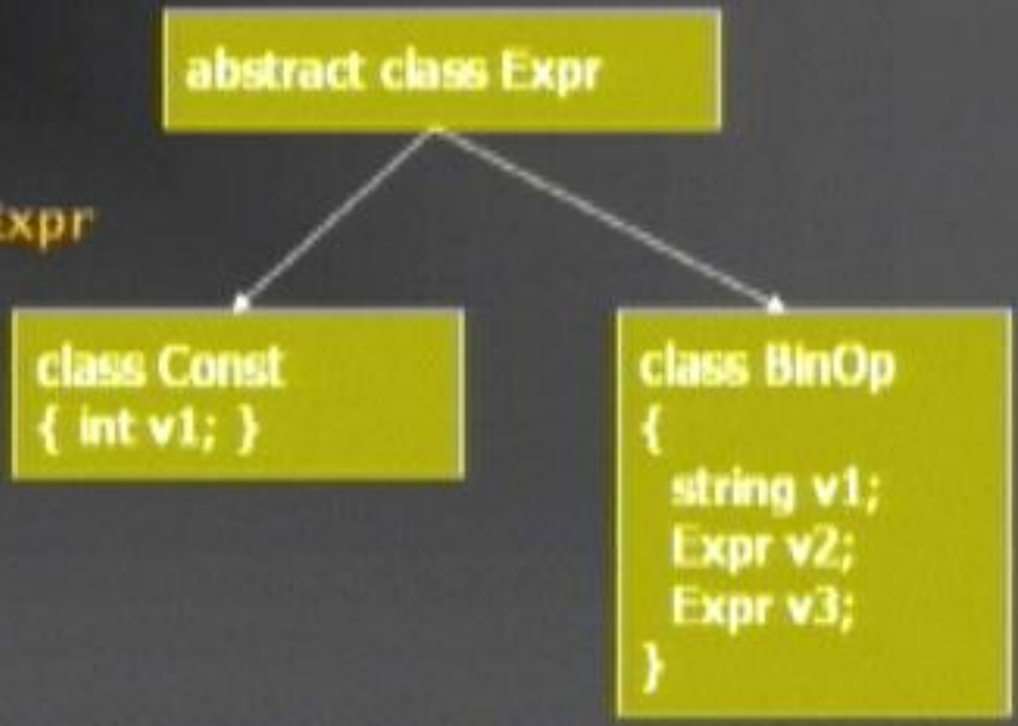
```
fun f (x,y) = …
datatype T = C1 of int*int | C2 of int
```

- Each tuple type generates a new class. Therefore try to share classes e.g. `int*string` `string*int` `{x:string,y:int}`

Compiling ML datatypes, cont.

- For all other datatypes, we could use the "inheritance idiom" e.g.

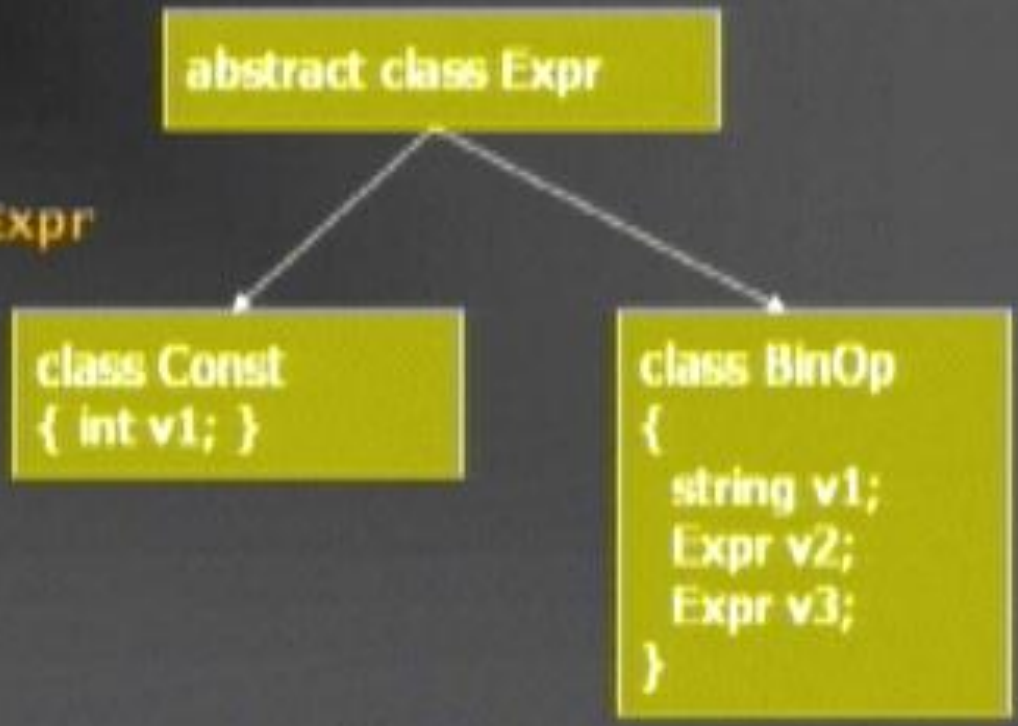
```
datatype Expr =
  Const of int
| BinOp of string*Expr*Expr
```



Compiling ML datatypes, cont.

- For all other datatypes, we could use the "inheritance idiom" e.g.

```
datatype Expr =
  Const of int
| BinOp of string*Expr*Expr
```



Compiling ML functions

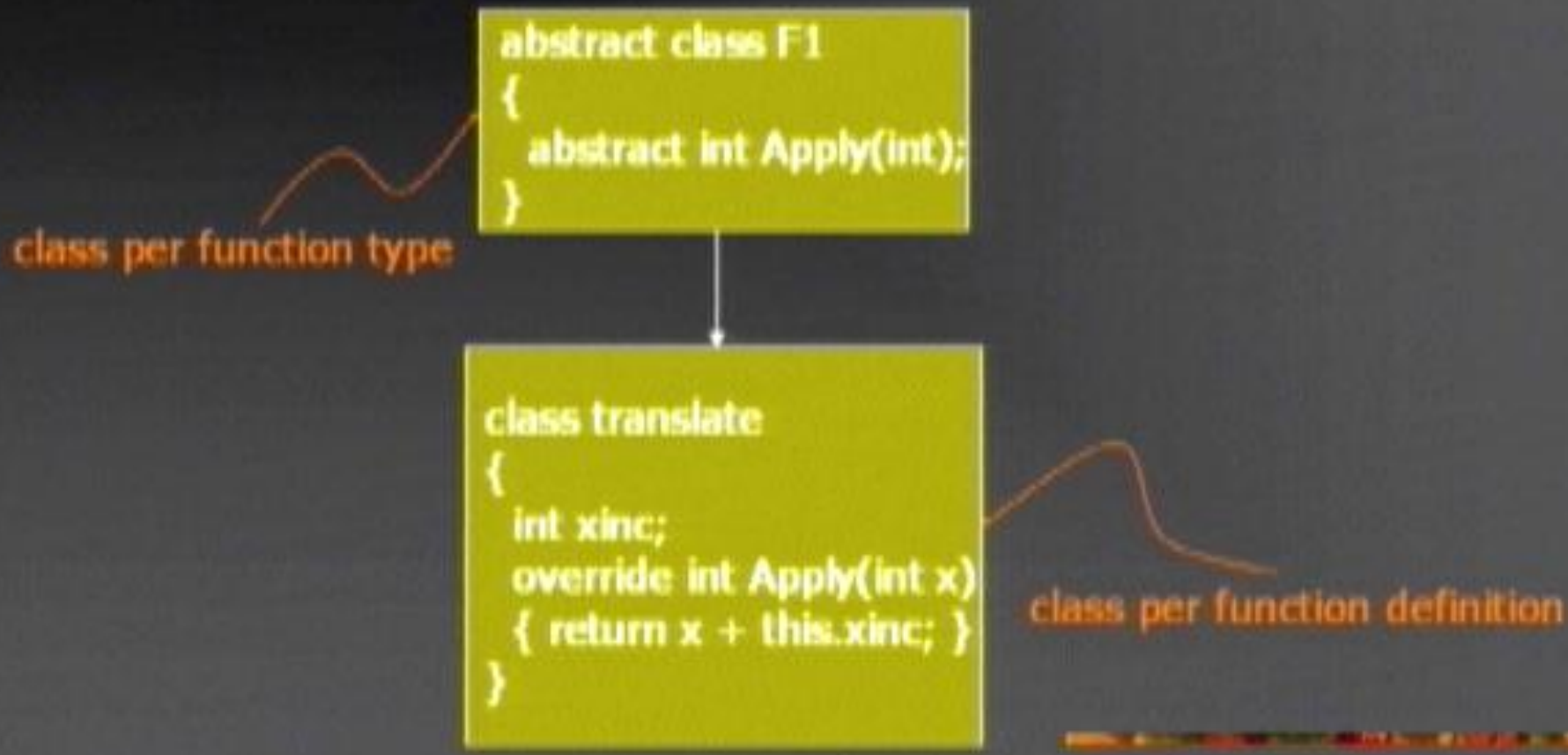
- When function isn't "first-class" just generate a static method or sometimes just a basic block
- Otherwise, we want a *closure*: a pair of
  - code pointer
  - environment (the free variables of the function)
- Example:

```
let fun translate x = x+xinc
in map translate coordlist end
```

Need to construct closure for `translate` with environment containing `xinc`

Compiling ML functions, cont.

- We could use the "inheritance idiom":



Compiling ML functions, cont.

- Downside: one class for every function type and one class for every function definition in the program.
- Could use delegates
  - Inefficient in CLR V1; also requires class per function type and class per environment type
- Instead, we play tricks to share classes (multiple apply methods per class)
- Better would be:
  - closures in the runtime
  - but just having generics is a big help…

Compiling ML polymorphism

- We could use the “inheritance idiom”:
  - uniform representation for values: System.Object
  - values with primitive must be "boxed"
- Drawback is performance:
  - boxing = heap allocation
  - run-time type-casts required (always succeed, but required for verification)

Compiling ML polymorphism, cont.

- Alternative used in current compiler: *specialise*.

  In theory: exponential code blowup.
  In practice: it doesn't happen. Why?

  - We specialise with respect to CLR representation e.g.
    `(length [2], length [Red])`
    generates only one version of length.
  - Specialisation leads to further optimisations.
  - Polymorphic functions tend to be small

Future: polymorphism in the CLR

- CLR V2 will support polymorphism (generics) directly
- This makes it *much* easier to compile SML, not only polymorphism but also
  - tuple types e.g. class Tuple3<A,B,C> { A v1; B v2; C v3; }
  - function types and closures e.g.

```
abstract class Fun<A,B> { abstract B Apply(A x); }
class Clos5 : Fun<int,int> {
  int fv1;
  override int Apply(int x) { return x+fv1; }
}
```

- Using generics we can almost support separate compilation…
  - but not of functors…
  - …and there is no CLR support for abstract types

Overview of talk

- Introduction
  - What is SML?
  - What is SML.NET?
- Language interop in SML.NET
- Demo
- Implementation
- Visual Studio integration

Recent work: VS integration

- SML.NET inside VS
  - syntax colouring & brace matching
  - automatic completion
  - debugger support (breakpoints, call stack, local vars)
  - continuous parsing & type inference
  - project support
- Implementation
  - Bootstrap the compiler to produce a .NET component for parsing, typechecking, etc. of SML
  - Use interlanguage working extensions to expose that as a COM component which can be glued into Visual Studio
  - Write new ML code to handle syntax highlighting, tooltips, error reporting, etc.

Recent work: VS integration

- SML.NET inside VS
  - syntax colouring & brace matching
  - automatic completion
  - debugger support (breakpoints, call stack, local vars)
  - continuous parsing & type inference
  - project support
- Implementation
  - Bootstrap the compiler to produce a .NET component for parsing, typechecking, etc. of SML
  - Use interlanguage working extensions to expose that as a COM component which can be glued into Visual Studio
  - Write new ML code to handle syntax highlighting, tooltips, error reporting, etc.

Server - Microsoft Visual C++ [design] - Server.aml

File Edit View Project Build Debug Tools Window Help

Server.aml

```
        (* finds coords which occur exactly 3 times in coordlist *)
        let fun f xover x3 x2 x1 [] = diff x3 xover
              | f xover x3 x2 x1 (a::x) =
                if member xover a then f xover x3 x2 x1 x else
                if member x3 a then f (a::xover) x3 x2 x1 x else
                if member x2 a then f xover (a::x3) x2 x1 x else
                if member x1 a then f xover x3 (a::x2) x1 x else
                                    f xover(x3 x2 (a::x1) x
            and diff x y = filter (not o member y) x
        in f [] [] [] [] x end
    in
     abstype generation = GEN of (int*int) list
       with
         fun alive (GEN livecoords) = livecoords
         and unsafegen c = GEN c
         and mkgen coordlist = GEN (lexordset coordlist)
         and nb_nextgen fn neighbours gen =
             let val living = alive gen
                 val isalive = member living
                 val liveneighbours = length o filter isalive o neighbours
                 fun twoorthree n = n=2 orelse n=3
(*BREAK*)        val survivors = filter (twoorthree o liveneighbours) living
(*BREAK*)        val newcoordlist = collect (filter (not o isalive) o neighbours) living
                 val newborn = occurs3 newcoordlist
             in union survivors (newborn) end
```

Solution Explorer - Server

Solution 'Demo' (2 projects)
- Client
  - References
  - App.ico
  - AssemblyInfo.cs
  - Client.cs
- Server
  - Source Files
    - Classes.aml
    - Rand.aml
    - Server.aml
    - script.smlnet
  - Readme.txt

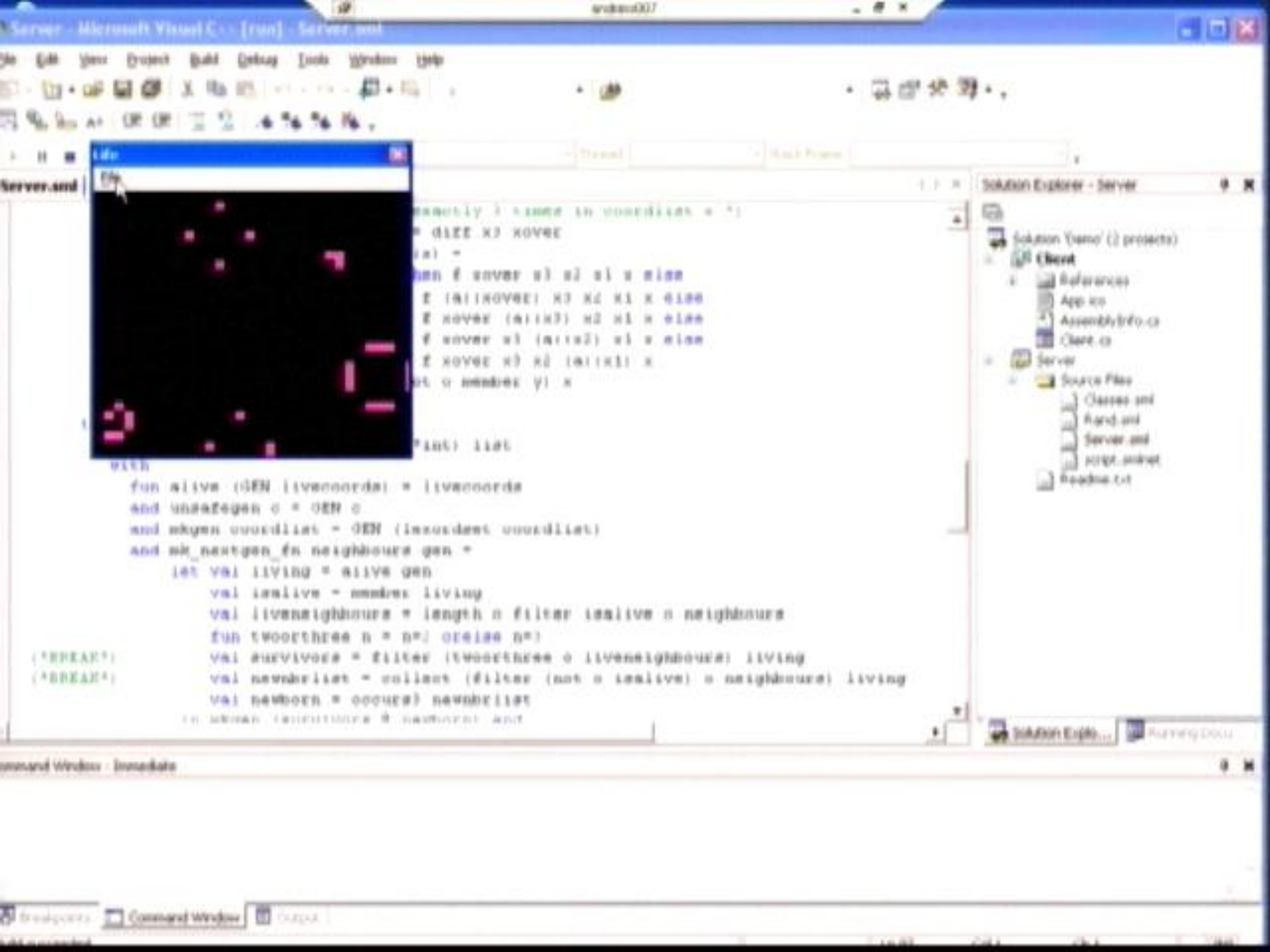
Solution Explorer | Class View

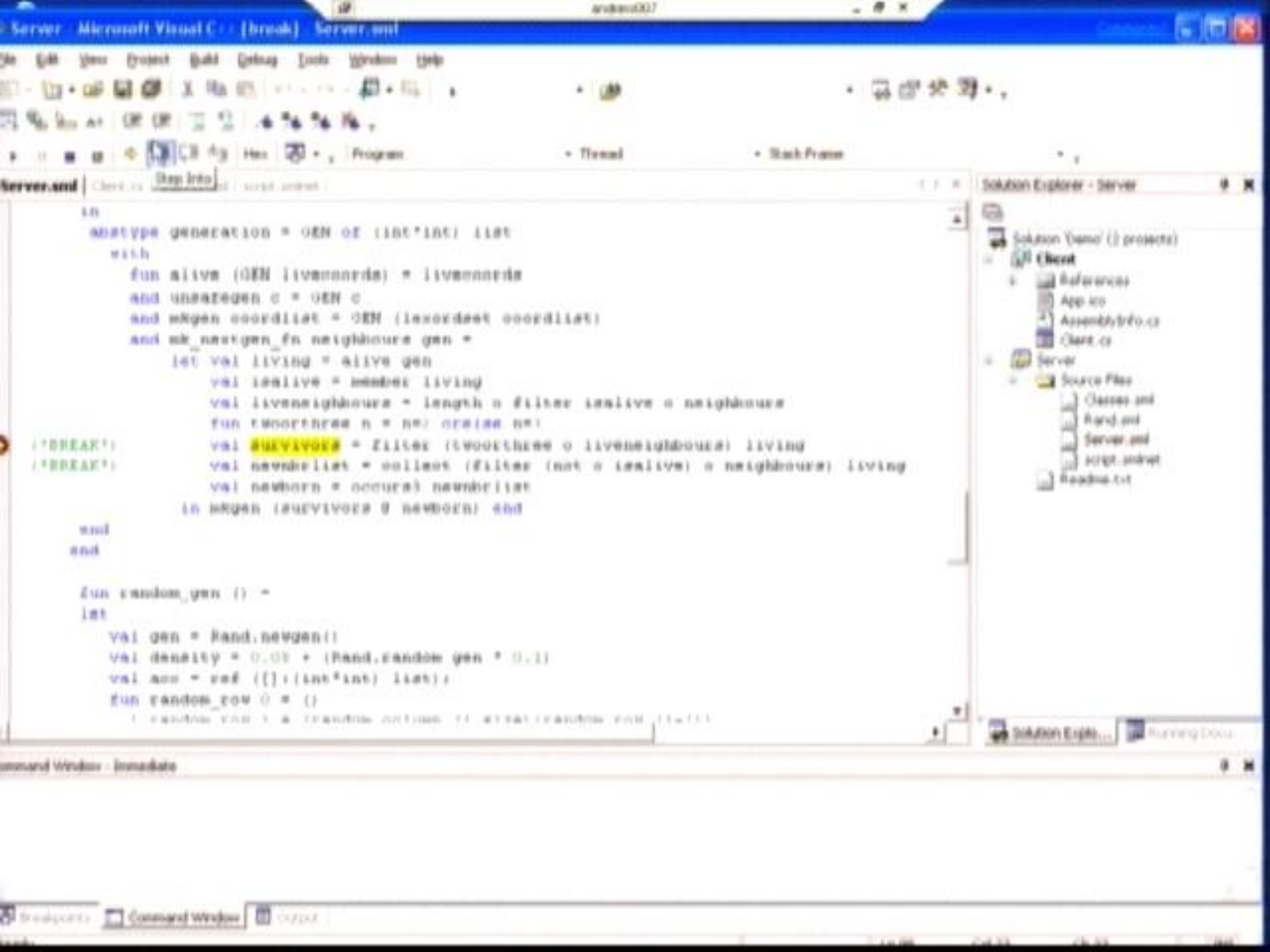
Properties

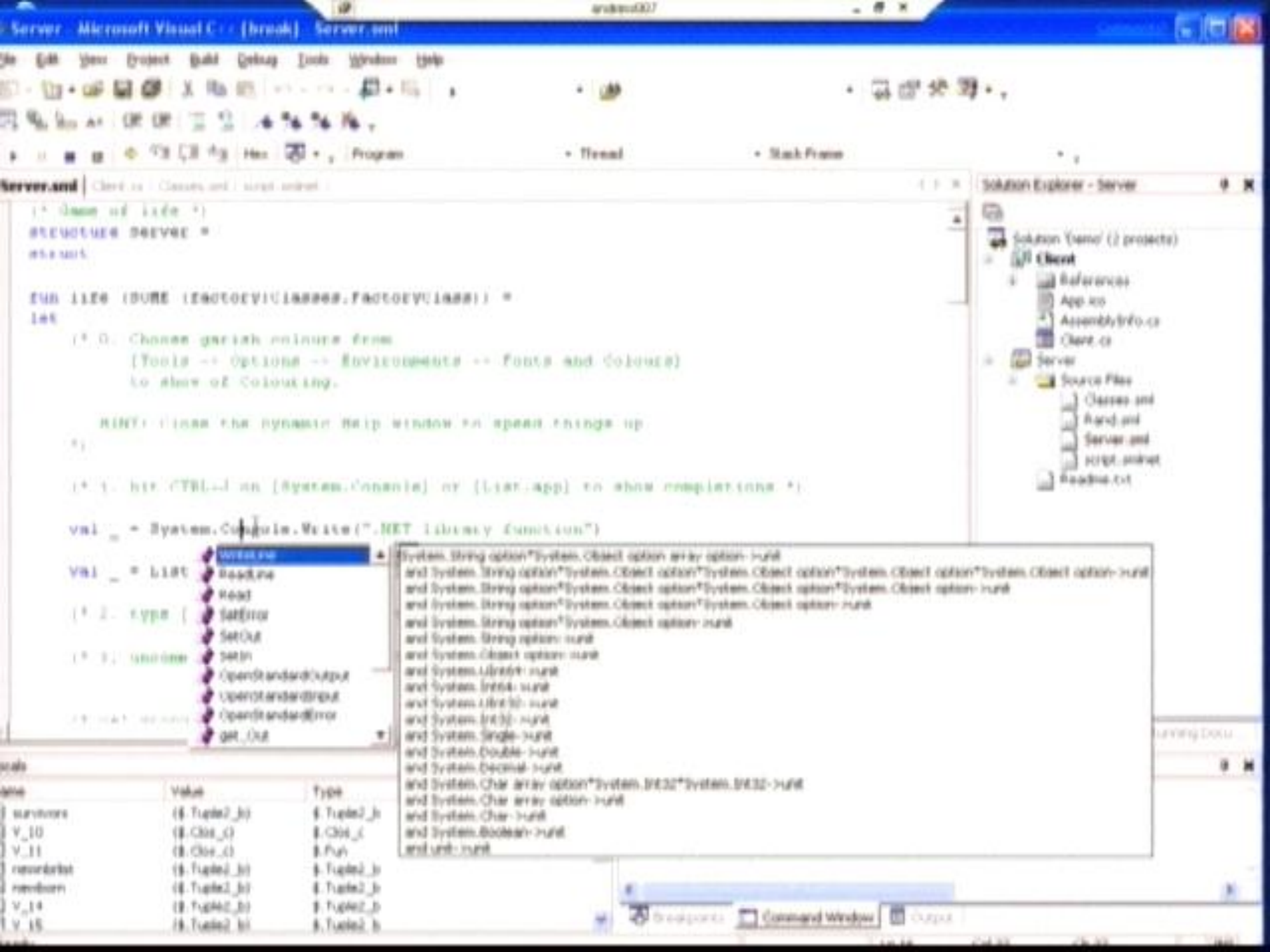
Output

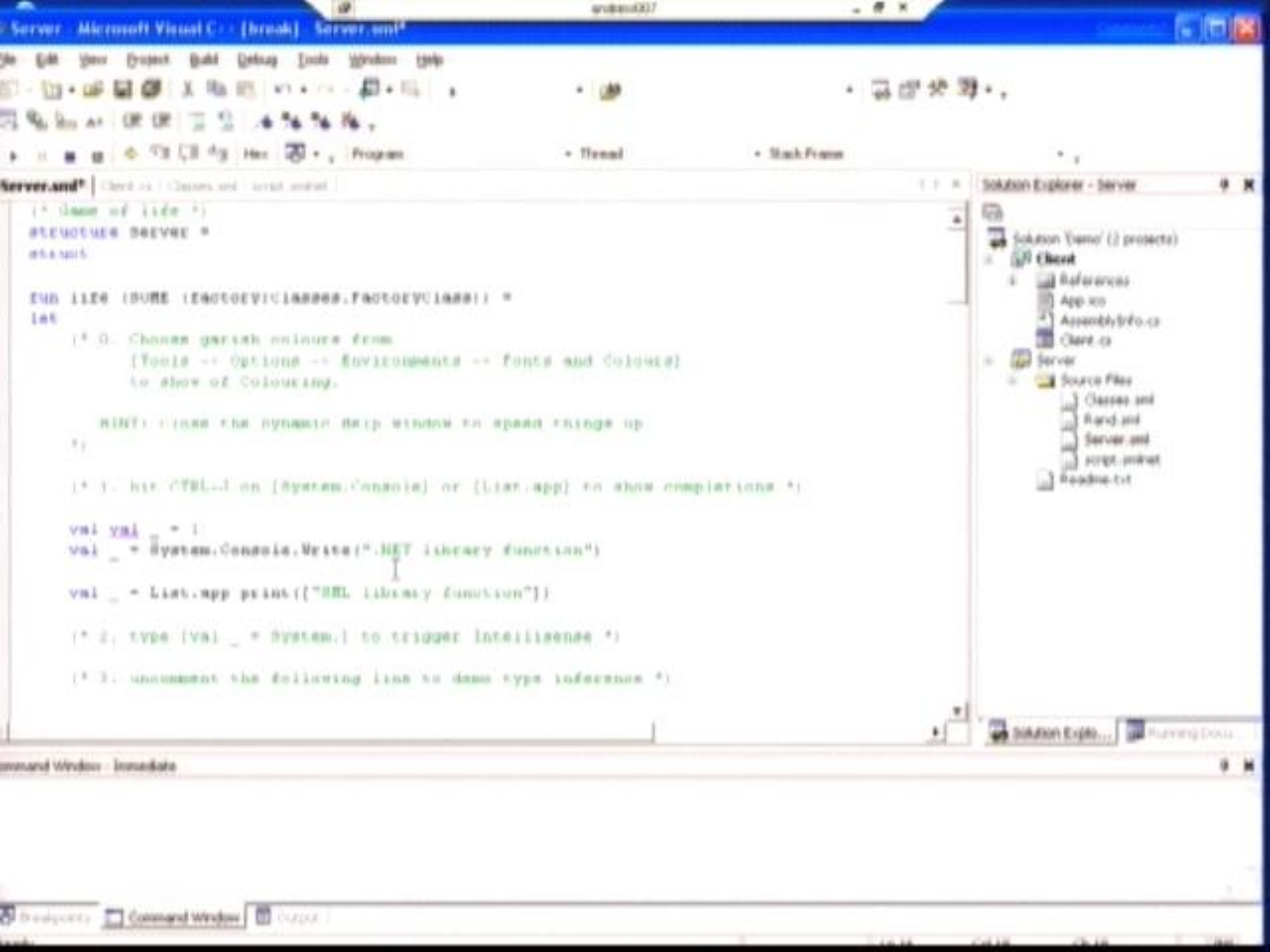
Task List | Output

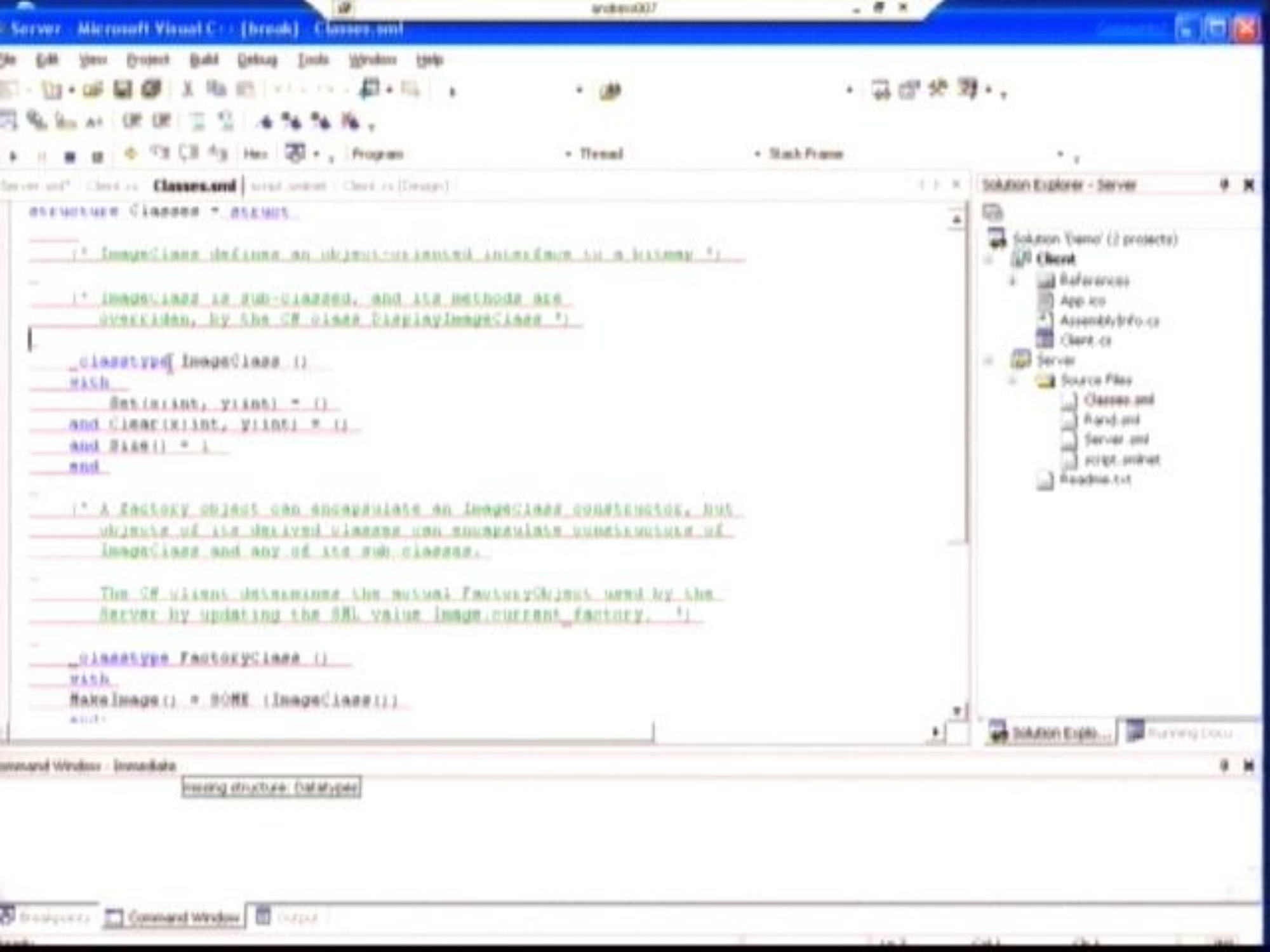
Properties | Dynamic Help


Server - Microsoft Visual C++ [run] - Server.aml
Life
Server.aml
Solution Explorer - Server
Solution 'Game' (2 projects)
Client
References
App.ico
AssemblyInfo.cs
Client.cs
Server
Source Files
Classes.aml
Rand.aml
Server.aml
script.aslrept
Readme.txt
Command Window - Immediate
Command Window


Server - Microsoft Visual C++ [break] Server.sml
File Edit View Project Build Debug Tools Window Help
Server.sml
Solution Explorer - Server
Solution 'Demo' (2 projects)
Client
References
App.ico
AssemblyInfo.cs
Client.cs
Server
Source Files
Classes.sml
Rand.sml
Server.sml
script.sml~
Readme.txt
Command Window - Immediate
Breakpoints
Command Window
Output


Server - Microsoft Visual C++ [break] Server.smt
Server.smt
(* Game of Life *)
structure Server =
fun life (world : (FactoryUi8888.FactoryUi888)) =
let
(* 0. Choose garish colours from
[Tools -> Options -> Environment -> Fonts and Colours]
to show of Colouring.
HINT: Close the dynamic help window to speed things up
*)
(* 1. hit CTRL-J on [System.Console] or [List.app] to show completions *)
val _ = System.Console.Write(".NET library function")
val _ = List
(* 2. type
(* 3. uncomm
Solution Explorer - Server
Solution 'demo' (2 projects)
Client
References
App.ico
AssemblyInfo.cs
Client.cs
Server
Source Files
Classes.sml
Rand.sml
Server.sml
script.sml.bat
Readme.txt
WriteLine
ReadLine
Read
SetError
SetOut
SetIn
OpenStandardOutput
OpenStandardInput
OpenStandardError
get_Out
System.String option*System.Object option array option->unit
and System.String option*System.Object option*System.Object option*System.Object option*System.Object option->unit
and System.String option*System.Object option*System.Object option*System.Object option->unit
and System.String option*System.Object option*System.Object option->unit
and System.String option*System.Object option->unit
and System.String option->unit
and System.Object option->unit
and System.UInt64->unit
and System.Int64->unit
and System.UInt32->unit
and System.Int32->unit
and System.Single->unit
and System.Double->unit
and System.Decimal->unit
and System.Char array option*System.Int32*System.Int32->unit
and System.Char array option->unit
and System.Char->unit
and System.Boolean->unit
and unit->unit
Value
Type
survivors
V_10
V_11
neverdoit
newborn
V_14
V_15
Command Window


Server - Microsoft Visual C++ [break] - Server.aml*
File Edit View Project Build Debug Tools Window Help
Server.aml*
(* Game of life *)
structure SERVER =
struct
fun life (SOME (FACTORYClasses.FactoryClass)) =
let
(* 0. Change parish colours from
[Tools -> Options -> Environments -> Fonts and Colours]
to show of Colouring.
HINT: close the dynamic help window to speed things up
*)
(* 1. hit 'TAB..' on [System.Console] or [List.app] to show completions *)
val val _ = 1
val _ = System.Console.Write(".NET library function")
val _ = List.app print["SML library function"]
(* 2. type [val _ = System.] to trigger Intellisense *)
(* 3. uncomment the following line to demo type inference *)
Solution Explorer - Server
Solution 'Demo' (2 projects)
Client
References
App.ico
AssemblyInfo.cs
Client.cs
Server
Source Files
Classes.aml
Rand.aml
Server.aml
script.aml.mlt
Readme.txt
Command Window - Immediate
Command Window


Server - Microsoft Visual C++ [break] - Classes.sml
File Edit View Project Build Debug Tools Window Help
Classes.sml
structure Classes = struct
(* ImageClass defines an object-oriented interface to a bitmap *)
(* ImageClass is sub-classed, and its methods are
overridden, by the C# class DisplayImageClass *)
classtype ImageClass ()
with
Set(xint, yint) = ()
and Clear(xint, yint) = ()
and Size() = ()
end
(* A factory object can encapsulate an ImageClass constructor, but
objects of its derived classes can encapsulate constructors of
ImageClass and any of its sub classes.
The C# client determines the actual FactoryObject used by the
Server by updating the SML value Image.current_factory. *)
classtype FactoryClass ()
with
MakeImage() = SOME (ImageClass())
end
Solution Explorer - Server
Solution 'Demo' (2 projects)
Client
References
App.ico
AssemblyInfo.cs
Client.cs
Server
Source Files
Classes.sml
Rand.sml
Server.sml
script.sml.sml
Readme.txt
Command Window - Immediate
Missing structure: DataTypes
Command Window